Laminar Unsteady Navier-Stokes Flow on Multicore Architectures

Bahareh Mostafazadeh, Aparna Chandramowlishwaran
Department of Electrical Engineering and Computer Science
University of California, Irvine
Email: bmostafa@uci.edu, amowli@uci.edu

Ferran Marti, Feng Liu
Department of Mechanical and Aerospace Engineering
University of California, Irvine
Email: fmarti@uci.edu, fliu@uci.edu

Abstract—Stencil computations are at the core of many scientific applications, and given that the fact that these computations are often memory-bound, they eventually become the limiting portion of these applications. Such limitations give rise to the increasing focus on optimizing stencil computations. A vast number of improvements have been applied to stencil kernels but very few of these attempts have applied the optimized stencils in real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications. Our work is unique in the sense that we target not just a stencil kernel for optimization, but an entire solver that captures the interaction between multiple stencil patterns and one that is capable of simulating real applications. Our implementation solves the fluid motion of compressible viscous and one that is capable of simulating real applications.

I. INTRODUCTION

Computational Fluid Dynamics (CFD) is a popular tool in engineering applications, and an active research area that is currently bound by computational resources. Grids which contain a large number of points (due to high accuracy of the problem, or due to a large problem domain), or require a high temporal resolution to capture high frequency motions have not been solved yet. Considering the computational capacities of today’s supercomputers, such simulations should have been possible, yet most applications developed in this area are not scalable to these supercomputers. This work aims to develop a more efficient and scalable solver to help overcome these obstacles and eventually enable new simulations in this field.

II. NUMERICAL METHODS

What we have implemented in this work is a solver for the Navier-Stokes equations based on ParCAE [5]. These equations are discretized in a Finite Volume Method with a cell-centered scheme and structured grid.

The spatial discretization for cell $I, J, K$ in 3D is,

$$\frac{\partial (\vec{W} \Omega)}{\partial t} |_{I,J,K} = - \sum_{m=1}^{N_F} ([\vec{F}_c - \vec{F}_v]_m \vec{n}_m \Delta S_m |_{I,J,K})$$

(1)

with $N_F$ equal to the number of cell faces (6 for a 3D hexahedral grid). The right hand side of Equation 1 is usually known as the residual.

We use the JST Scheme [4] to calculate the convective fluxes $(\vec{F}_c)$. On face $I + 1/2, J, K$, the convective flux is

$$\vec{F}_c \vec{n} \Delta S_{I+1/2,J,K} \approx \vec{F}_c \vec{n} \Delta S_{I+1/2,J,K}$$

(2)

where $\vec{n}$ is the artificial dissipation, and

$$\vec{W}_{I+1/2,J,K} = \frac{1}{2} (\vec{W}_{I,J,K} + \vec{W}_{I+1,J,K})$$

(3)

The dissipative term is a blend between second and fourth order differences. It uses a pressure sensor to determine proximity to a shock wave to activate strong artificial dissipation, and it uses minimal dissipation otherwise just to avoid numerical oscillations. According to Equation 3, to calculate the viscous flux at each cell, we need to read one neighboring cell for each surface resulting in a 7 point stencil for a 3D grid. Similarly, updating the artificial dissipation term for each cell leads to a 13 point 3D stencil. Considering these stencil kernels, our solver has a high bandwidth-to-compute ratio which limits the scalability of this application.

III. OUR IMPLEMENTATION

As a first step, we implement the laminar unsteady flow solver in C++. The code iteratively updates the entire grid until we reach a convergence threshold, and at each iteration, we run 5 stages of Runge-Kutta. Each stage mainly consists of the flux calculations followed by updating the values of $\vec{W}$ and Pressure for each cell of the grid and finally updating the boundary conditions. We apply the following optimizations to improve the performance of our CFD solver.

A. Single Core Optimizations

First, we focus on improving the sequential implementation of our solver. We applied a number of single-core optimizations including finding the most efficient loop structures by unrolling, jamming, interchanging the loops, and also replacing expensive math operations with cheaper operations. Due to the low arithmetic intensity of this application, one important optimization is to reduce the cache misses. This is critical since for a grid with a considerable size, the last level cache will not be able to store the entire grid. We thus break the grid into smaller blocks with tunable size that could
be fit into the last level cache, and iterate on each of these blocks.

### B. Vectorization

We have used the Intel C++ compiler to reach a fully vectorized implementation by using the following transformations.

- **Loop Unswitching**: This results in single-entry loops, which the compiler is able to vectorize.
- **Loop Unrolling**: Unrolling small inner loops results in vectorization of loops with more iterations.
- **Loop fission**: Dividing big loops with dependencies into smaller independent loops helps the compiler vectorize.

### C. Parallelization

Using OpenMP for parallelization, we divide either the entire grid, or one block after last level cache blocking into equal chunks, and assign the calculation of each chunk to one thread. This decomposition is shown in Figure 1.

![Fig. 1](image)

Fig. 1: The domain is decomposed into two levels to utilize L1 (green blocks) and last level cache (orange blocks). Each red block is assigned to one thread.

We need to consider the effects of NUMA to scale with increasing number of cores. Assuming a first touch placement policy, we use the OpenMP binding policies to parallelize the initialization so that the data each core will later access is also initialized by the same core and on the same socket. Avoiding false sharing among multiple threads is also necessary in order to achieve scalability, especially since it could potentially increase as the number of threads grows. To avoid false sharing, we first pad the array so that each thread works on a chunk size that is a multiple of the cache line size. This way, one thread does not invalidate another threads data in cache. We also maximize the usage of private data for each thread and store values such as the fluxes in local data structures for each thread.

### IV. Results and Discussion

We evaluate our implementation on a dual-socket Intel Xeon E5-2630 v3 processor with 8 core per socket and peak performance of 614.4 GFlops for single precision and 1228.8 GFlops for double precision. Each core has an L1 and L2 cache of 32KB and 256 KB respectively and all the cores on a socket share 20 MB of L3 cache.

To show the improvement of each optimization, we illustrate the GFlops achieved for different number of threads for a step flow simulation on a grid with 500000 cells in Figure 2. It is worth noting that the initial implementation had achieved 2.18 GFlops for single precision and 1.28 GFlops for double precision and our single threaded optimized implementation shows more than $5 \times$ improvement compared to the baseline. Furthermore, our optimized OpenMP implementation shows 1.8, 3.2, 5.52 and $8.85 \times$ speedup with 2, 4, 8 and 16 threads respectively.

![Fig. 2](image)

Fig. 2: GFlops improvement per optimization for different number of threads. These results were obtained from a step flow simulation on a grid with 50000 cells.

### V. Conclusion

This work presents an optimized solver that maps well onto multicore architectures and is a building-block for a future scalable software infrastructure for next-generation CFD. While some of the optimizations discussed in this work have been previously applied to stencil computations [1], [2], [6], [3], this is one of the first efforts in addressing the challenges of an entire solver which consists of multiple stencils along with an in-depth analysis and optimization of an important real-world application.

### REFERENCES


